You ain't dead yet.
We want something new!
But now we can!
I present you
The Micro Frontends! 🎉
Ivan Jovanovic
Senior software engineer @ nearForm
https://ivanjov.com
@ivanjov96
What are Micro Frontends?
Micro Frontends - a microservice approach to the modern web
What are Microservices?
Microservices is an architectural style that structures an application as a collection of loosely coupled services.
Event bus
Event bus allows publish-subscribe-style communication between components without requiring the components to explicitly register with one another.
Microservice principles

• Lightweight protocol between services
• Small services, one job per service
• Service independence
• Easier to understand, develop and test
• Speeds up development
• Enables continues delivery and deployment
Diagram showing three JavaScript (JS) logos connected with arrows, indicating a network or relationship.
MICROSERVICES

MICROSERVICES EVERYWHERE
How did we get here?

- Monolith full stack apps 😊
- Frontend and backend apps 😄
- Microservices in the backend 😅
- Components on the UI 😎
- Micro frontends 😍
We need to fight with the bad architecture!
What problem does it solve?

• Use new frontend framework on the old architecture
• No more shared codebases and conflicts
• Independent deployments
• Each team can pick their own stack
• You can easily scale your frontend app
How to split apps?

• By functionality on the page
• By page
• By section
Implementation

• External app bootstrapping
• IFrames
• Single-spa library
• Frint framework
External app bootstrapping
External app bootstrapping

• Code lives on different server
• Independent deployment
• Communication is done through:
  • Window object
  • Event bus
```javascript
const fetchApp = (element, baseUrl, files = ['main.js', 'main.css']) => {
    const fnName = element.toLowerCase()
                .replace(/^[^\.#]*\./, '')

    files.forEach((filename) => {
        const fileUrl = baseUrl + '/' + filename

        if (.js$.test(filename)) {
            $.ajax(
                {dataType: 'script',
                cache: true,
                url: fileUrl
            }).done(() => {
                global[fnName](element)
            })
        }

        if (.css$.test(filename)) {
            $('<link/>', {
                rel: 'stylesheet',
                type: 'text/css',
                href: fileUrl
            }).appendTo('head')
        }
    })
}
```
files.forEach((filename) => {
  const fileUrl = baseUrl + '/' + filename

  if (.js$.test(filename)) {
    $.ajax(
      {dataType: 'script',
       cache: true,
       url: fileUrl
      }).done(() => {
        global[fnName](element)
      })
  }

  if (.css$.test(filename)) {
    $('</link />', {
      rel: 'stylesheet',
      type: 'text/css',
      href: fileUrl
    }).appendTo('head')
  }
})
import React from 'react'
import { render } from 'react-dom'

export default (el) => {
  render(
    <h1>Hello</h1>,
    document.querySelector(el))
}
React app export

```javascript
1 import App from './main'
2
3 global['app'] = App
```
Fetching React app in the Root app

```javascript
import fetchApp from './fetch-app'
fetchApp({
el: '#app',
baseUrl: 'http://localhost:3000'
})
```
Event bus - Eev

- [https://github.com/chrisdavies/eev](https://github.com/chrisdavies/eev)
- Less than 500 bytes minified + zipped
- Really fast
- Zero dependencies
- Simple
A tiny, fast, zero-dependency event emitter for JavaScript.
import Eev from 'eev'

const e = new Eev()

e.emit('event', { foo: 'Bar' })

e.on('event', function (data) {
  console.log('got ' + data.foo) // got bar
})
import Eev from 'eav'

window.e = new Eev()

window.e.emit('event', { foo: 'Bar' })

window.e.on('event', function (data) {
  console.log('got ' + data.foo) // got bar
})
IFrames
IFrames

- Code lives on different server
- Independent deployment
- Communication is done through browser “Event bus”
window.postMessage(data, origin)
window.addEventListener('message', receiveMessage)
<html>
<head>
<title>IFrame services</title>
<script type='text/javascript'>
const app = window.getElementById('app-iframe').contentWindow
app.postMessage('This will not work', 'https://google.com') // url doesn’t match
app.postMessage('This will work, hi there!', 'http://example.com')
function receiveMessage (event) {
  if (event.origin !== 'http://example.com') // security check for the origin
    return
  console.log(event.source) // iframe
  console.log(event.data) // 'hi from an iframe'
}
window.addEventListener('message', receiveMessage)
</script>
</head>
<body>
<div id='app'>
<iframe src="https://example.com/app.html" id='app-iframe'></iframe>
</div>
</body>
</html>
const app = window.getElementById('app-iframe').contentWindow

app.postMessage('This will not work', 'https://google.com') // url doesn't match

app.postMessage('This will work, hi there!', 'http://example.com')

function receiveMessage(event) {
  if (event.origin !== 'http://example.com') // security check for the origin
    return

  console.log(event.source) // iframe
  console.log(event.data) // 'hi from an iframe'
}

window.addEventListener('message', receiveMessage)
```javascript
function receiveMessage (event) {
  if (event.origin !== "http://example.com")
    return

  console.log(event.source) // window.opener
  console.log(event.data) // 'This will work, hi there!'

  event.source.postMessage('hi from an iframe', event.origin)
}

window.addEventListener('message', receiveMessage)
```
Single-spa library
Single-spa library

- https://github.com/CanopyTax/single-spa
- Use multiple frameworks on the same page without refreshing the page
- Write code using a new framework, without rewriting your existing app
- Lazy load code for improved initial load time
The Javascript metaframework

- 309 commits
- 5 branches
- 72 releases
- 13 contributors

- master
- New pull request
  - Find file
  - Clone or download

- 2038: Update README.md
- fixed path to Ecosystem file
- Added link to single-spa-portal in examples README
- Pass reference to single-spa as a prop to lifecycle functions
- Documentation
- Rewriting tests to use karma-webpack only. Removing systemjs/spm
- Implementing #78. First mount events (#101)
- Rewrite for 2.0
- Adding travis config for saucelabs
- License and readme
- Update README.md
- Implementing #78. First mount events (#101)
- Upgrading browser versions
- Polyfilling promises for IE11
- v3.9.1
- Rewriting tests to use karma-webpack only. Removing systemjs/spm
Single-spa library

• Code lives on the same server

• Everything is bundled and deployed at the same time

• Communication is done through:
  • Window object
  • Event bus
  • Shared state (Redux etc.)
  • Whatever works for you…
import * as singleSpa from 'single-spa'

const appName = 'app1'

const loadingFunction = () => import('./app1/app1.js')

const activityFunction = location => location.hash.startsWith('#/app1')

singleSpa.declareChildApplication(appName, loadingFunction, activityFunction)

singleSpa.start()
const loadingFunction = () => import('./app1/app1.js')
Activity function

```javascript
const activityFunction = location => location.hash.startsWith('#/ap1')
```
1 singleSpa.declareChildApplication(appName, loadingFunction, activityFunction)
2
3 singleSpa.start()
Single-spa app

```javascript
1 export const bootstrap (props) => {}
2
3 export const mount (props) => {}  
4
5 export const unmount (props) => {}
```
let domEl

// Make a div for your app
export const bootstrap = (props) => {
  return Promise
    .resolve()
    .then(() => {
      domEl = document.createElement('div')
      domEl.id = 'app1'
      document.body.appendChild(domEl)
    })
}
1 // Mount the framework code
2 export const mount = (props) => {
3     return Promise
4         .resolve()
5         .then(() => {
6             domEl.textContent = 'App 1 is mounted!'
7         })
8     }

```javascript
// Unmount the spa framework from the dom
export const unmount = (props) => {
  return Promise
    .resolve()
    .then(() => {
      domEl.textContent = ''
    })
}
```
import React from 'react'
import ReactDOM from 'react-dom'
import rootComponent from './path-to-root-component'
import singleSpaReact from 'single-spa-react'

const reactLifecycles = singleSpaReact({
  React,
  ReactDOM,
  rootComponent,
  domElementGetter: () => document.getElementById('main-content')
})

export const bootstrap = [
  reactLifecycles.bootstrap
]

export const mount = [
  reactLifecycles.mount
]

export const unmount = [
  reactLifecycles.unmount
]
```
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic'
import singleSpaAngular2 from 'single-spa-angular2'
import mainModule from './main-module.ts'
import { Router } from '@angular/router'

const ng2Lifecycles = singleSpaAngular2({
  domElementGetter: () => document.getElementById('angular2'),
  mainModule,
  angularPlatform: platformBrowserDynamic(),
  template: `<component-to-render />`,
  Router
})

export const bootstrap = [
  ng2Lifecycles.bootstrap
]

export const mount = [
  ng2Lifecycles.mount
]

export const unmount = [
  ng2Lifecycles.unmount
]
```
```javascript
import Vue from 'vue'
import singleSpaVue from 'single-spa-vue'

const vueLifecycles = singleSpaVue({
  Vue,
  appOptions: {
    el: '#mount-location',
    template: '<div>some template</div>'
  }
})

export const bootstrap = [
  vueLifecycles.bootstrap
]

export const mount = [
  vueLifecycles.mount
]

export const unmount = [
  vueLifecycles.unmount
]
```
Print
Frint

• https://frint.js.org/

• CLI

• Routing

• State management

• Server side rendering supported
Frint CLI

$ npm install -g frint-cli
$ print new my-directory --example kitchensink
Reactive Programming 🎉
It works with React Native!!!
Who is using Micro Frontends?
Google
Microsoft
Facebook
And many more...
Everyone! 😎
Conclusion

• Don’t use this if you have a simple app

• Use micro frontends to make things easier, not complicated

• Micro frontends architecture doesn’t mean to use every framework in the world

• Don’t forget to make standards across micro apps
Thank you! 😊